# R&D Project : TFHE

Tilio PILET Baptiste THIERRY Nicolas MENDEL-BOUCHARIN

May 2025

## Contents

# 1 Introduction

The simplest definition of homomorphic encryption is for all plaintexts $m_1$ and $m_2$, $c_1$ and $c_2$ their associated ciphertexts, we have that

1. $c_1 + c_2 = E_k(m_1 + m_2)$
2. $c_1 \cdot c_2 = E_k(m_1 \times m_2)$

Below, we will study a better definition and examples of such encryption schemes, but for now, it can be noted that some of the well-known encryption schemes have one of those properties.

## 1.1 Partially homomorphic encryption

If an encryption scheme has one the properties aboves it is said to be partially homomorphic.

**Examples** Quick recall on RSA : Let $p$ and $q$ be two prime numbers, $d \in \mathbb{N}$ such that $\gcd(d, (p-1)(q-1)) = 1$ and $e \in \mathbb{N}$ such that $e.d = 1 \mod (p-1)(q-1)$. We define $n = pq$ and $\varphi(n) = (p-1)(q-1)$. The private key is $(p, q, d)$ and the public key is $(n, e)$. The cipher of a message $m$ is $c \equiv m^e \pmod{n}$. We can easily verify that $c^d \equiv m^{e^d} \equiv m^{ed} \equiv m \pmod{n}$. This encryption is partially homomorphic. Let $m_1, m_2, c_1 \equiv m_1^e \mod n, c_2 \equiv m_2^e \mod n$. We have : $(c_1 c_2)^d \equiv (m_1^e * m_2^e)^d \equiv (m_1 * m_2)^{e^d} \equiv m_1 m_2 \pmod{n}$

Another partially homomorphic encyrption is El Gamal : In a cyclic group $G$ of order $q$ with generator $g$, if the public key is $(G, q, g, h)$, where $h = g^x$, and $x$ is the secret key, then the encryption of a message $m$ is $\mathcal{E}(m) = (g^r, m \cdot h^r)$, for some random $r \in \{0, \ldots, q-1\}$. The homomorphic property is then: $\mathcal{E}(m_1) \cdot \mathcal{E}(m_2) = (g^{r_1}, m_1 \cdot h^{r_1})(g^{r_2}, m_2 \cdot h^{r_2}) = (g^{r_1+r_2}, (m_1 \cdot m_2)h^{r_1+r_2}) = \mathcal{E}(m_1 \cdot m_2)$

## 1.2 Somewhat Homomorphic Encryption

Some encryptions schemes are said to be "somewhat homomorphic". It means that you can do an arbitrary number of homomorphic operations. They support both additions and multiplications but each of them add "noise" (2.4) so much that at some point the ciphertext is undecipherable. There are a lot of those schemes but before TFHE and bootstrapping the number of operations was limited.

## 1.3 Lattice-based cryptography

**Definition 1.1** (Lattice)**.** Let $E$ a normed vector space over $\mathbb{R}$ of finite dimension $n$. Let $L \subset E$. $L$ is called a lattice if $L = \mathbb{Z}e_1 + ... + \mathbb{Z}e_n$ where $(e_1, ..., e_n)$ is a basis of $E$.

Lattices can be used for cryptography because we can take advantage of hard lattice problems, for instance :

- The Shortest Vector Problem (SVP): it consists in finding the shortest non zero vector in a given lattice
- The $\gamma$-approximate Shortest Vector Problem (SVP$_\gamma$): identify a vector that is almost the shortest vector; formally, identify $v \in L$ such that $||v|| \leq \gamma \times \lambda$ where $\lambda = min\{||v||; v \in L\}$
- The Decisional Shortest Vector Problem (GAPSVP$\gamma, r$): given $\gamma \geq 1$ and $r > 0$, decide if $\lambda \leq r$ or $\lambda \geq \gamma.r$ (where $\lambda$ is defined as in SVP

# 2 Fully Homomorphic Encryption (FHE)

## 2.1 Definition

**Definition 2.1** (Homomorphic Encryption (HE))**.** Let $C$ the ciphertext space, $M$ the message space, and $F$ the evaluation function space. We name homomorphic encryption the algorithm quadruplet $(\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathsf{Eval})$ such as :

- $\mathcal{K} : \mathbb{N} \to PK \times SK$ is the public key and secret key generation algorithm
- $\mathcal{E} : \mathcal{M} \to \mathcal{C}$ is the encryption algorithm
- $\mathcal{D} : \mathcal{C} \to \mathcal{M}$ is the decryption algorithm
- $\mathsf{Eval} : F \times \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ the evaluation function

This cryptosystem is homomorphic if : $\forall f \in F,\ \mathcal{D}(\mathsf{Eval}(f, C_1, C_2)) = \mathsf{Eval}(f, \mathcal{D}(C_1), \mathcal{D}(C_2)))$

**Definition 2.2** (Fully Homomorphic Encryption (FHE))**.** An homomorphic encryption scheme is said to be Fully Homomorphic if

- the set F of evaluable functions is the set of all efficiently-computable functions
- the ciphertext growth at function evaluation doesn't depend on the complexity of the function evaluated

**Remark:** An SHE(Somewhat Homomorphic Encryption) scheme is an HE scheme that is not fully homomorphic.

## 2.2 History

FHE is a new field of cryptography; the first plausible construction was made in 2009 by Craig Gentry. His scheme used lattices and was supporting additions and multiplications on ciphertext. Although it started as a somewhat homomorphic scheme he made it fully homomorphic with bootstrapping (we will elaborate on this later). One of the main issues that was restraining somewhat homomorphic schemes was that each operation was adding noise, but Gentry found a way to correct this with bootstrapping.

In 2011-2012 some new techniques were developed by Zvika Brakerski, Craig Gentry, Vinox Vaikunanathan, and others. They led to much more efficient FHE following the basic Gentry's original construction. Most of those new algorithm security was based on the hardness of the (Ring) Learning With Errors (We will follow up on that later).

In 2013, Craig Gentry, Amit Sahai, and Brent Waters created a new FHE scheme to avoid a step of "linearization." They also found and improved a new bootstrapping technique to create the TFHE cryptosystem that we study.

From the original Gentry's scheme, the idea remained the same but new algorithms were created to lower the cost and more importantly the noise.

## 2.3 Why do we want to have Fully Homomorphic Encryption ?

Fully Homomorphic Encryption would enable a server to process encrypted sensitive data for some clients without learning anything about this data. It could guarantee to users of cloud services/artificial intelligences that their data will remain private. Storing encrypted data online could also prevent companies from training their AIs on someone's work or sensitive data. Here is a short list of possible applications of FHE : online key-value database, privacy-preserving machine learning, statistics over confidential data...

## 2.4 Noise

Most solutions for FHE rely on hard lattice problems. To ensure a good security of the encryption, the ciphertexts must contain some added 'noise'. Unfortunately each homomorphic operation adds some noise. The noise can accumulate and grow enough to overflow the data making the decryption impossible. That's why we only had somewhat HE until quite recently. The first scheme to overcome this problem and TFHE are both using bootstrapping.

## 2.5 Bootstrapping

Let $K_1$ the private key and $K_2$ another key that we will use for bootstrapping, called bootstrapping key. Let's say we have a ciphertext $C = \mathcal{E}_{K_1}(m)$ and want to reduce its noise. We compute $C_b = \mathcal{E}_{K_2}(C)$ and $K_b = \mathcal{E}_{K_2}(K_1)$. We evaluate the decryption function with encrypted parameters $C_b$ and $K_b$ (the key can be seen as a parameter). Then, thanks to the homomorphic properties we obtain : $\mathcal{D}_{K_b}(C_b) = \mathcal{D}_{\mathcal{E}_{K_2}(K_1)}(\mathcal{E}_{K_2}(C)) = \mathcal{E}_{K_2}(\mathcal{D}_{K_1}(C)) = \mathcal{E}_{K_2}(m)$ where the second equality is not strictly true but is there to show that both sides have the same decryption under $K_2$. We obtained a new ciphertext for $m$ but under $K_2$. With good parameters, it can be shown that this operation will reduce the noise.

**Remark:** For a public-key FHE, $K_2$ can be the public key. We call circular security the assumption that making public the encryption of the private key with the public key is safe.

## 2.6 LWE, RLWE and GLWE

### 2.6.1 Learning With Errors (LWE)

This problem and its decisional version play a key role in today's lattice-based cryptography. They were introduced by Regev in 2005.

**Definition 2.3** (LWE)**.** Let $n \in \mathbb{N}_{>0}$, and $q \in \mathbb{N}_{>1}$. Let $\chi$ a gaussian distribution over $\mathbb{Z}_q$ and $s \in \mathbb{Z}_q^n$

An LWE sample is a tuple of the form $(a_1, a_2, ..., a_n, b) \in \mathbb{Z}_q^{n+1}$ where $b =< a, s > +e \ mod \ q = \sum_{i=1}^{n} a_i.s_i + e \ mod \ q$ with :

- $a = (a_1, a_2, ..., a_n)$ taken following a uniform distribution
- $e \in \mathbb{Z}_q$ taken following $\chi$

Given arbitrarily many independent LWE samples, the problem consists in finding back s. The decisional version consists in distinguishing LWE samples from random uniform samples $(a_1, a_2, ..., a_n, a_{n+1}) \in \mathbb{Z}_q^{n+1}$.

In 2009 Peikert provided a reduction from GAPSVP problem to LWE. This reduction and many others (quantum reductions, reductions of worst-case lattice problems to a certain case of LWE, ...) show that the LWE problem is at least as hard as well-known hard lattice problems.

### 2.6.2 Ring Learning With Errors (RLWE)

The RLWE problem is the ring version of LWE. Specifically, while LWE is in $\mathbb{Z}_q^{n+1}$, RLWE takes place in $R_q^2$ where $R_q = \mathbb{Z}_q[x]/ < f(x) >$ where $f(x) \in \mathbb{Z}_q[x]$ is a monic irreducible polynomial of degree $d$ and now $q$ is a prime.

**Definition 2.4** (RLWE)**.** The RLWE problem is to discover $s \in R_q$ given access to arbitrarily many independent samples $(a, b = s.a + e) \in R_q \times R_q$ where a is chosen uniformly at random in $R_q$ and $e \in R_q$ is sampled from an error distribution $\chi$. The decisional version of this problem consists in distinguishing uniformly random samples in $R_q$ from RLWE samples.

Like LWE, researchers showed that RLWE is as hard as hard lattice problems.

### 2.6.3 Global Learning With Errors (GLWE)

The GLWE problem is a generalisation of the LWE problem and the RLWE problem

**Definition 2.5** (GLWE)**.** The GLWE problem consists in finding $s/in R_q^k$ given a list of noisy equations from :

$\{(a, b =< a, s > +e)\} \in R_q^k \times R_q$, where $a \xleftarrow{\mathcal{U}} R_q^k, e \xleftarrow{\mathcal{U}} \chi\}$

For $R = \mathbb{Z}$ we have LWE and for $k = 1$ we have RLWE. This problem was proven to be hard and is a pillar of the TFHE security

# 3 TFHE

## 3.1 Advantages and drawbacks of this type of FHE

TFHE [5] has a very fast bootstrapping operation, way better than other FHE schemes. By using the Fastest Fourier Transform in the West (or upgrades), bootstrapping can be performed in less than a second. TFHE is also good for bit-wise operations i.e., when computations are expressed as boolean circuits. Another (non-negligible) advantage of TFHE is that it has programmable-bootstrapping, allowing the evaluation of non-linear functions via lookup tables while bootstrapping.

TFHE's main limitation is the lack of support for batching, where batching is a technique that allows processing several messages simultaneously (can be done by putting them in vectors for example). In some situations this can make TFHE less competitive than BGV/BHV or CKKS. While TFHE is better for bit-wise operations, it is outperformed by CKKS for operations over real numbers, and probably also by BGV/BHV for operations over integers.[1]

## 3.2 Definitions

### 3.2.1 Torus

**Definition 3.1** (Torus)**.** The torus $\mathbb{T}$ is defined by $\mathbb{R}/\mathbb{Z} = [0,1) + \mathbb{Z}$. It's an abelian group, so it has a structure of $\mathbb{Z}$-module, but this is not a ring because the multiplication is not defined.

Let's take an example of why multiplication is not defined. If we take $a = b = \frac{1}{4}$ and $c = \frac{1}{3}$ then we have $(a+b) \times c = \frac{1}{2} \times \frac{1}{3} = \frac{1}{6}$ and $a \times c + b \times c = \frac{1}{8} + \frac{1}{8} = \frac{1}{4}$. The problem comes from the fact that in this group $0 = 1$. We can then define polynomials over the torus :

**Definition 3.2** (Torus polynomials)**.** Let $\Phi(X)$ denote the unique irreducible polynomial with integer coefficients that divides $X^M - 1$ but not $X^k - 1$ for $k < M$ and let $N$ denote its degree. Considering the polynomial ring $\mathbb{T}_N[X] = \mathbb{R}[X]/\mathbb{Z}_N[X] = \mathbb{T}[X]/(X^N + 1)$

For performance reasons, $M$ is a power of 2 and then $N = M/2$. Elements of $\mathbb{T}_N[X]$ are polynomials modulo $X^N + 1$ with coefficient in $\mathbb{T}$. But being also a $\mathbb{Z}_N[X]$-module, its elements can be added together and multiplied by polynomials of $\mathbb{Z}_N[X]$.

### 3.2.2 Discretized torus

Let $B$ be an integer $\geq 2$, we can represent any number $t$ of the Torus $\mathbb{T}$ by an infinite sequence of number $(t_1, t_2, \ldots)$ with $t_i \in \{0, \ldots, B-1\}$ and we have $t = \sum_{i \geq 1} \frac{t_i}{B^i}$. If we fix the length of the sequence, let's call it $w$, then we get the representation $t = \sum_{i=1}^{w} \frac{t_i}{B^i}$. This representation limits the torus to the subset $\frac{1}{B^w}\mathbb{Z}/\mathbb{Z} \subset \mathbb{T}$ with representatives in $\{0, \frac{1}{B^w}, \frac{2}{B^w}, \ldots, \frac{B^w-1}{B^w}\}$.

If we take $B = 2$, then we obtain the coefficients of the sequence are bits, i.e, $t_i \in \{0,1\}$ and $t = \sum_{i=1}^{w} \frac{t_i}{2^i}$. Thus, every element can be seen as a real number in $[-\frac{1}{2}, \frac{1}{2})$ or as an unsigned real number in $[0,1)$. We end up with the following definition

**Definition 3.3** (Discretized torus)**.** The discretized torus $\mathbb{T}_q$ is the set $\{\frac{i}{q} \mod 1 | i \in \mathbb{Z}\} = \{\frac{i}{q} | i \in \mathbb{Z}/q\mathbb{Z}\} = \{0, \frac{1}{q}, \ldots, \frac{q-1}{q}\}$ with $q = 2^w$ and $w \in \mathbb{N}$, $w \geq 3$.

This discretized torus can be identified with $\mathbb{Z}/q\mathbb{Z}$ and the computation on $\mathbb{T}_q$ can be done only in $\mathbb{Z}/q\mathbb{Z}$ with the numerator of the fraction.

As we do it for the torus $\mathbb{T}$, we can define

$$\mathbb{T}_{N,q}[X] = \mathbb{T}_q[X]/ < X^N + 1 >$$

We also define $Z_{N,q}[X] = \mathbb{Z}_q[X]/ < X^N + 1 >$ where $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$. If we see $\frac{1}{q}$ as an element of $\mathbb{T}_{N,q}[X]$ then every element $P \in \mathbb{T}_{N,q}[X]$ can be expressed as $P = \frac{1}{q}\bar{P}$ for some polynomial $\bar{P} \in Z_{N,q}[X]$.

## 3.3 Notations

Before going into the subject, we denote

- $\lambda$ the security parameter.
- $a \xleftarrow{\mathcal{U}} A$ the fact that $a$ as been sample uniformly at random in $A$ with $A$ a set.
- $a \leftarrow \mathcal{D}$ the fact that $a$ as been sample according to $\mathcal{D}$ with $\mathcal{D}$ a probability distribution.
- $\lfloor x \rceil$ is the nearest integer to $x$.
- $N$ is a power of two.
- $\mathbb{B}$ is the set $\{0, 1\}$.
- $\mathbb{B}_N[X]$ is the subset of polynomials in $\mathbb{Z}_N[X]$ with binary coefficients.
- $< a, b >$ is the standard scalar product with $a$ and $b$ two vectors.

## 3.4 Redefinition of LWE and GLWE for TFHE

**Definition 3.4** (LWE problem over the discretized Torus)**.** Let $q, n \in \mathbb{N}$ and $s = (s_1, ..., s_n) \xleftarrow{\mathcal{U}} \mathbb{B}^n$. Let also $\hat{\chi}$ be an error distribution over $\frac{1}{q}\mathbb{Z}$. The learning with errors over the discretized torus problem is to distinguish samples chosen according to the following distributions:

$$\mathcal{D}_0 = \{(a, r) | a \xleftarrow{\mathcal{U}} \mathbb{T}_q^n, r \xleftarrow{\mathcal{U}} \mathbb{T}_q\}$$

and

$$\mathcal{D}_1 = \{(a, r) | a = (a_1, ..., a_n) \xleftarrow{\mathcal{U}} \mathbb{T}_q^n, r = \sum_{j=1}^{n} s_j \cdot a_j + e, e \leftarrow \hat{\chi}\}$$

**Definition 3.5** (GLWE problem over the discretized torus)**.** Let $N, q, k \in \mathbb{N}$ with $N$ a power of 2 and let $s = (s_1, ..., s_k) \xleftarrow{\mathcal{U}} \mathbb{B}_N[X]^k$. Let also $\hat{\chi}$ be an error distribution over $\frac{1}{q}\mathbb{Z}_N[X]$; namely, over polynomials of $q^{-1}\mathbb{Z}_N[X]$ with coefficients drawn according to $\hat{\chi}$. The general LWE problem over the discretized torus is to distinguish samples chosen according to the following distributions:

$$\mathcal{D}_0 = \{(a, r) | a \xleftarrow{\mathcal{U}} \mathbb{T}_{N,q}[X]^k, r \xleftarrow{\mathcal{U}} \mathbb{T}_{N,q}[X]\}$$

and

$$\mathcal{D}_1 = \{(a, r) | a = (a_1, ..., a_n) \xleftarrow{\mathcal{U}} \mathbb{T}_{N,q}[X]^k, r = \sum_{j=1}^{n} s_j \cdot a_j + e, e \leftarrow \hat{\chi}\}$$

The decisional LWE assumption(2.6.1) (resp. the decisional GLWE assumption(2.6.3)) asserts that solving the LWE problem (resp. GLWE problem) is infeasible for some security parameter $\lambda$, where $q, n, \hat{\chi}$ (resp. $N, q, k, \hat{\chi}$) are functions of $\lambda$.

## 3.5 TLWE

Based on the LWE problem (2.6.1), we want to construct an encryption scheme over the discretized torus $\mathbb{T}_q$. We first remark that an element $r$ of $\mathbb{T}_q$ of the form $r = \sum_{i=1}^{n} s_j \cdot a_j + e$ can't be distinguished from a random element of $\mathbb{T}_q$ thanks to the LWE assumption. So, we use $r = \sum_{i=1}^{n} s_j \cdot a_j + e$ to create the ciphertext of a message $\mu \in \mathbb{T}_q$ and we get $\mathbf{c} = (a_1, \ldots, a_n, r + \mu) \in \mathbb{T}_q^{n+1}$, where $s = (s_1, \ldots, s_n) \in \mathbb{B}^n$ is the private key. The secret key $s$ is in $\mathbb{B}^n$ because it permits an efficient implementation of the bootstrapping.

Now, we will see after that to get a unique description if the noise is not too large, we need to restrict the set of plaintexts as $\mathcal{P} \subsetneq \mathbb{T}_q$ an additive group. In fact, we take $\mathcal{P} = \{0, \frac{1}{p}, \ldots, \frac{p-1}{p}\} = \mathbb{T}_p$ with $p|q$ and $p \geq 2$.

### 3.5.1 TLWE Encryption

By the idea said before, we get the following encryption scheme over LWE

**KeyGen($\lambda$) :** Using the security paramter $\lambda$, define

- $n \in \mathbb{N}^+$.
- $p, q \in \mathbb{N}^+$ such that $p|q$.
- a discretized error distribution $\hat{\chi}$ on $\frac{1}{q}\mathbb{Z}$ induced by a normal distribution $\chi = \mathcal{N}(0, \sigma^2)$ on $\mathbb{R}$.
- $s \xleftarrow{\mathcal{U}} \mathbb{B}^n$ the private key.

Then the space of plaintexts is $\mathcal{P} = \mathbb{T}_p \subsetneq \mathbb{T}_q$, the public parameters are $\text{pp} = (n, \sigma, p, q)$ and the private key is $\text{sk} = s$.

**Encyrption$_{sk}(\mu)$ :** The encryption of $m \in \mathcal{P}$ is given by

$$\mathbf{c} = \text{TLWE}_s(\mu) = (a_1, \dots, a_n, b) \in \mathbb{T}_q^{n+1}$$

with $b = \sum_{i=1}^n s_i \cdot a_j + \mu + e =< s, a > +\mu + e$ where $(a_1, \dots, a_n) \xleftarrow{\mathcal{U}} \mathbb{T}_q^n$ and a "small" noise $e \leftarrow \hat{\chi}$.

**Decryption$_{sk}(\mu)$ :** Given a ciphertext $\mathbf{c} = (a_1, \dots, a_n, b) \in \mathbb{T}_q^{n+1}$, we start by computing

$$\mu^* = b - \sum_{i=1}^n s_i \cdot a_i$$

and next, return

$$\mu = \frac{\lfloor p\mu^* \rceil \mod p}{p}$$

as the decryption of $\mathbf{c}$, that correspond to the closet plaintext $\mu \in \mathcal{P}$.

We will now see that the decryption returns the correct plaintext $\mu \in \mathcal{P}$ if the noise $e$ satisfies $|e| < \frac{1}{2p}$.

*Proof.* For $\mu \in \mathcal{P} = \{0, \frac{1}{p}, \dots, \frac{p-1}{p}\}$, let $\mathbf{c} = \text{TLWE}(\mu) = (a_1, \dots, a_n, b)$ where $(a_1, \dots, a_n) \xleftarrow{\mathcal{U}} \mathbb{T}_q^n$ and $b = \sum_{i=0}^n s_i \cdot a_i + e$ with $e \leftarrow \hat{\chi}$. Since $\mu \in \mathcal{P}$, $\exists! \mu' \in [\![0, p-1]\!]$ such that $\mu = \frac{\mu'}{p}$. With the notation of the Decryption function, we got

$$\mu^* = b - \sum_{i=0}^n a_i \cdot s_i = \mu + e \mod 1 = \mu + e + k \text{ with } k \in \mathbb{Z}$$

Thus,

$$\lfloor p\mu^* \rceil = \lfloor p(\mu + e) + kp \rceil = \lfloor p(\mu + e) \rceil + kp$$

and

$$\lfloor p(\mu + e) \rceil = \lfloor p(\frac{\mu'}{p} + e) \rceil = \lfloor \mu' + pe \rceil \underbrace{=\mu'}_{\text{if } |e| < \frac{1}{2p}}$$

Finally, we get

$$\frac{\lfloor p\mu^* \rceil \mod p}{p} = \frac{\mu'}{p} = \mu$$

□

### 3.5.2 Encoding and Decoding

In the last paragraph, we saw that we can encrypt and decrypt elements over the discretized torus. The remaining question is: How to encode an element in the discretized torus? In this section, we will detail some encoding and decoding methods for bits and elements in $\mathbb{Z}_p$. The function Encode is applied before encryption and Decode is applied after decryption.

**Bits** If we want to encode bits in the discretized torus, one can use the function Encode$(b) = \frac{b}{2}$, this function maps 0 on $\frac{0}{q} \in \mathbb{T}_q$ and 1 on $\frac{1}{2} = \frac{q/2}{q} \in \mathbb{T}_q$. The reverse operation is defined as Decode$(\mu) = \lfloor 2 * \mu \rceil \mod 2$, so if $\mu \in \{0, \frac{1}{2}\}$ then Decode$(\mu) \in \{0, 1\}$.

**Elements in $\mathbb{Z}_p$** We can generalize this encoding of bits into the encoding of integers modulo $p$ (bits are integers modulo $p = 2$). To do so we define Encode$(k) = \frac{(k \mod p)\Delta}{q}$ with $\Delta = \frac{p}{q}$ and Decode$(\mu) = \lfloor p\mu \rceil \mod p$.

## 3.6 TGLWE

Here we replace operations on the torus $\mathbb{T}_q$ by operations on polynomials of degree $\leq N$ with coefficients in $\mathbb{T}_q$. The plaintext space becomes $\mathcal{P}_N[X] := \mathcal{P}[X]/(X^N + 1) = \mathbb{T}_{N,p}[X] \subset \mathbb{T}_{N,q}[X]$.

### 3.6.1 TGLWE Encryption

**Keygen($\lambda$):** In order to create a key for a chosen security parameter $\lambda$ proceed as follows:

- define a number of bits $N$ and $k \geq 1$.
- choose $p$ and $q$ such as $p|q$.
- generate a discretisized error distribution vector induced by a normal distribution $\chi = \mathcal{N}(0, \sigma^2)$ : $\hat{\chi}$
- use it to sample uniformly a vector$s = (s_1, \ldots, s_k) \leftarrow \mathbb{B}_N[X]^k$

The public parameters are now $(k, N, \sigma, p, q)$ and the private key is just $s_k = s$

**Encrypt$_{sk}(\mu)$:** Using the private key given by the previous algorithm to encrypt a plaintext $\mu \in \mathcal{P}_N[X]$ proceeds as follows :

- generate a random vector $(a_1, \ldots, a_k) \leftarrow \mathbb{T}_{N,q}[X]^k$
- using $\hat{\chi}$ generate a small noise $e$
- compute $b = \sum_{j=1}^{k} s_j \cdot a_j + \mu$

The encryption of $\mu$ is given by :
$c \leftarrow \text{TGLWE}_s\mu = (a_1, \ldots, a_k, b) \in \mathbb{T}_{N,q}[X]^{k+1}$

**Decrypt$_{sk}$(c):** To decrypt $c = (a_1, \ldots, a_k, b)$ using the private key s you can compute:
$\mu^* = b - \sum_{j=1}^{k} s_j \cdot a_j$
Then you have to find the closest plaintext $\mu \in \mathcal{P}_N[x]$
We already observed that TLWE is a TGLWE encryption with parameters $(k, N) = (n, 1)$. Even so, both encryptions are needed for the bootstrapping. Still, TLWE should be preferred for encryption of a single torus element $\mu \in \mathcal{P}$ because the resulting ciphertext is shorter.

### 3.6.2 Encoding and Decoding

As said just before, the TGLWE encryption scheme can work under a restriction to a polynomial of degree 0 seen as an element in $\mathcal{P}$. If we are in this case the encoding and decoding function are the same as in TLWE. When up to $N$ element in a torus need to be encrypted they can be seen as coefficient of polynomial $\mu(X) = \mu_0 + \mu_1 X + \cdots + \mu_{N-1} \in \mathcal{P}_N[X]$, this process is called coefficient packing.

## 3.7 Operations over Encrypted Data

### 3.7.1 TLWE Ciphertexts

**Addititions** Let $c_1 \leftarrow \text{TLWE}_s(\mu_1)$ and $c_2 \leftarrow \text{TLWE}_s(\mu_2)$ be respective encryptions of $\mu_1$ and $\mu_2$: We then have $c_1 = (a_1, \dots, a_n, b)$ ; $c_2 = (a'_1, \dots, a'_n, b')$ with $(a_1, \dots, a_n) \xleftarrow{\mathcal{U}} \mathbb{T}_q^n$ ; $b = \sum_{j=1}^n s_j \cdot a_j + \mu_1 + e_1$ and $(a'_1, \dots, a'_n) \xleftarrow{\mathcal{U}} \mathbb{T}_q^n$ ; $b' = \sum_{j=1}^n s_j \cdot a'_j + \mu_2 + e_2$. Assuming the errors $e_1, e_2$ are small then clearly $c_3 := c_1 + c_2 = (a_1 + a'_1, \dots, a_n + a'_n, b + b')$ is an TLWE encryption of $\mu_1 + \mu_2$.

**Multiplication by a known constant** Again, under the assumption that the noise keeps "small" we can multiply by a constant. To do so we just see it as a series of additions. $K \cdot c = c + \cdots + c$ ($K$ times). Then with $c = (a_1, \dots, a_n, b) \in \mathbb{T}_q^{n+1}$ we have : $K \cdot c = (K \cdot a_1, \dots, K \cdot a_n, K \cdot b)$

**Ciphertexts multiplications** In order to multiply ciphertexts, we first have to express the matrices with the "gadget decomposition" technique. The main idea of this decomposition is to decompose every entry of $\mathbf{c} = \text{TLWE}(\mu) \in \mathbb{T}_q^{n+1}$ with the same technique used to define the discretized torus. This decomposition will give us a new encryption scheme, with the plaintext in $\mathbb{Z}_p$, named TGSW and represented by matrices. This encryption will allow us to do homomorphic multiplication between two TGSW ciphertexts or a TGSW ciphertext and a TLWE ciphertext. To be more precise, we will define two homomorphic multiplications :

- Internal product $\boxtimes$ :

$$\begin{aligned} \boxtimes\text{: TGSW} \times \text{TGSW} &\to \text{TGSW} \\ (\text{TGSW}(m_1), \text{TGSW}(m_2)) &\mapsto \text{TGSW}(m_1) \boxtimes \text{TGSW}(m2) = \text{TGSW}(m_1 \cdot m_2) \end{aligned}$$

- External product $\boxdot$ :

$$\begin{aligned} \boxdot\text{: TGSW} \times \text{TLWE} &\to \text{TLWE} \\ (\text{TGSW}(m_1), \text{TLWE}(\mu_2)) &\mapsto \text{TGSW}(m_1) \boxdot \text{TGSW}(\mu_2) = \text{TLWE}(m_1 \cdot \mu_2) \end{aligned}$$

It's preferred to use the second one because the TLWE ciphertexts are shorter (vectors) than TGSW one (matrices). We can now start to define "gadget" matrices that will be central in the TGSW encryption.

**Definition 3.6** (Gadget Matrix)**.** Let $\mathbb{T}_q = \frac{1}{q}\mathbb{Z}/\mathbb{Z}$ the discretized torus for an integer $q$, $B$ a radix and $\ell \geq 1$ such that $B^\ell | q$ the "gadget matrix" $G \in \mathcal{M}_{n+1,(n+1)\ell}(\mathbb{T}_q)$ is defined as :

$$G^T := diag(g^T, \dots, g^T) = \begin{pmatrix} 1/B & & & & \\ \vdots & & & & \\ 1/B^\ell & & & & \\ & 1/B & & & \\ & \vdots & & & \\ & 1/B^\ell & & & \\ & & \ddots & & \\ & & & 1/B & \\ & & & \vdots & \\ & & & 1/B^\ell & \end{pmatrix}$$

with $g = (1/B, 1/B^2, \dots, 1/B^\ell) \in \mathbb{T}_q^\ell$

With such matrix, we can generate an element of the torus $\mathbb{T}_q^{n+1}$: for all $u \in \mathbb{Z}^{(n+1)\ell}$we have $u \cdot G^T \in \mathbb{T}_q^{n+1}$. Let $u = (u_1^{(1)}, \dots, u_l^{(1)}, u_1^{(2)}, \dots, u_l^{(2)}, \dots, u_1^{(n+1)}, \dots, u_l^{(n+1)}) \in \mathbb{Z}^{(n+1)\ell}$ then we get

$$u.G^T = \Big( \sum_{i=1}^{\ell} \frac{u_i^{(k)}}{B^i} \Big)_{k \in [\![1, n+1]\!]} = \Big( < u^{(k)}, g^T > \Big)_{k \in [\![1, n+1]\!]} \text{ with } u^{(k)} = (u_1^{(k)}, \dots, u_l^{(k)})$$

and, since every element of $\mathbb{T}_q$ has the form $\sum_{i=1}^{\ell} \frac{t_i}{B^i}$ then we have that $u \cdot G^T \in \mathbb{T}_q^{n+1}$.

We can construct the inverse transformation $G^{-1} : \mathbb{T}_q^{n+1} \to \mathbb{Z}^{(n+1)\ell}$ such that for $v \in \mathbb{T}_q^{n+1}$ we have $G^{-1}(v) \cdot G^T \approx v$. This inverse transformation replaces each entry of a vector by its signed radix-B expansion: if $v = (v_1, \ldots, v_{n+1}) \in \mathbb{T}_q^{n+1} \subset \mathbb{T}^n = ([-\frac{1}{2}; \frac{1}{2}) + \mathbb{Z})^n$ with $v_i \in [-\frac{1}{2}, \frac{1}{2})$ then we set $\bar{v_i} = \lfloor B^\ell v_i \rceil \in [\lfloor -\frac{B^\ell}{2} \rfloor, \lceil \frac{B^\ell}{2} \rceil)$ and write :

$$\bar{v_i} \equiv \sum_{j=1}^{\ell} u_{i,j} B^{\ell - j} \pmod{B^\ell} \text{ where } u_{i,j} \in [-\lfloor \tfrac{B}{2} \rfloor, \lceil \tfrac{B}{2} \rceil)$$

We can now define $g^{-1}(v_i) := (u_{i,1}, \ldots, u_{i,\ell}) \in \mathbb{Z}^\ell$ Then

$$\begin{aligned} \mathrm{G}^{-1}(v) :&= (g^{-1}(v_1), g^{-1}(v_2), \ldots, g^{-1}(v_{n+1})) \\ &= (u_{1,1}, \ldots, u_{1,\ell}, \ldots, u_{2,1}, \ldots, u_{2,\ell}, \ldots, u_{n+1,1}, \ldots, u_{n+1,\ell}) \in \mathbb{Z}^{(n+1)\ell} \end{aligned} \tag{1}$$

If $B^l = q$, then we got $v_i = \frac{v_i'}{q} \in \mathbb{T}_q$ with $v_i' \in [\![ \lfloor \frac{-(q-1)}{2} \rfloor, \lceil \frac{q-1}{2} \rceil ]\!]$ then $\bar{v_i} = B^l \cdot v_i = v_i'$. Thus, give us that $\mathrm{G}^{-1}(v) \cdot \mathrm{G}^T = v$ on $\mathbb{T}_q$.

**Remark :** We can extend the inverse transformation $\mathrm{G}^{-1}$ to matrices. For a matrix $M \in \mathcal{M}_{k,n+1}(\mathbb{T}_p)$, we will have $\mathrm{G}^{-1}(M)$ in $\mathcal{M}_{k,(n+1)\ell}(\mathbb{Z})$ and define by

$$\mathrm{G}^{-1}(M) = \left(\mathrm{G}^{-1}(M_i)\right)_{i=1,\ldots,k} \text{ where } M_i \text{ is the i-th line of } M$$

**TGSW Encryption** Now we want to construct TGSW encryption with the help of the gadget matrix. Let $q = 2^w$ and $p \in \mathbb{N}^+$ such that $p|q$. We assume that $B^\ell = p$ (so $B = 2^{w'}$), then $G$ is defined on $\mathbb{T}_p$. We now take $m \in \mathbb{Z}_p$ and $m \cdot G^T \in \mathcal{M}_{(n+1)\ell,n+1}(\mathbb{T}_p) \subset \mathcal{M}_{(n+1)\ell,n+1}(\mathbb{T}_q)$ gives the decomposition of $m$ in $\mathbb{T}_p$ on every columns. Now, we want to apply a mask to this matrix and we do so by using the matrix

$$Z = \begin{pmatrix} \mathrm{TLWE}_s(0) \\ \mathrm{TLWE}_s(0) \\ \vdots \\ \mathrm{TLWE}_s(0) \end{pmatrix} \in \mathcal{M}_{(n+1)\ell,n+1}(\mathbb{T}_q)$$

with $s \in \mathbb{B}^n$ a secret key. Then, we can now construct the ciphertext of $m$ defined by

$$\mathrm{TGSW}_s(m) = Z + m \cdot G^T = \begin{pmatrix} \mathrm{TLWE}_s(0) + (\frac{m}{B}, 0, \ldots, 0) \\ \mathrm{TLWE}_s(0) + (\frac{m}{B^2}, 0, \ldots, 0) \\ \vdots \\ \mathrm{TLWE}_s(0) + (\frac{m}{p}, 0, \ldots, 0) \\ \vdots \\ \mathrm{TLWE}_s(0) + (0, \ldots, 0, \frac{m}{B}) \\ \vdots \\ \mathrm{TLWE}_s(0) + (0, \ldots, 0, \frac{m}{p}) \end{pmatrix} \in \mathcal{M}_{(n+1)\ell,n+1}(\mathbb{T}_q)$$

**Remark :** The $l$ last rows of $\mathrm{TGSW}_s(m)$ are $(\mathrm{TLWE}_s(0) + (0, \ldots, 0, \frac{m}{B^i}))_{i \in [\![1, \log_2(p)]\!]}$ and this correspond to $\mathrm{TLWE}_s(\mu_i)$ with $\mu_i = \frac{m}{B^i} \in \mathbb{T}_p$. Then we can get a TLWE encryption of $\mu = \sum_{i=0}^{l} \mu_i$ under the secret key $s$ by summing the $l$ last rows of $\mathrm{TGSW}_s(m)$, indeed, we have

$$\begin{aligned} \sum_{i=0}^{l} \big(\mathrm{TLWE}_s(0) + (0, \ldots, 0, \mu_i)\big) &= \mathrm{TLWE}_s(0) + \sum_{i=0}^{l} (0, \ldots, 0, \mu_i) \\ &= \mathrm{TLWE}_s(0) + (0, \ldots, 0, \sum_{i=0}^{l} \mu_i) \\ &= \mathrm{TLWE}_s(0) + (0, \ldots, 0, \mu) \\ &= \mathrm{TLWE}_s(\mu) \end{aligned} \tag{2}$$

**Internal product :** The best improvement here is that since the plaintexts are defined on $\mathbb{Z}_p$, they can be multiplied and we can now define the internal product

$$\begin{aligned} \boxtimes\colon \mathrm{TGSW}_s \times \mathrm{TGSW}_s &\to \mathrm{TGSW}_s \\ (C1, C2) &\mapsto C1 \boxtimes C2 = G^{-1}(C2) \cdot C1 \end{aligned}$$

Let $m_1, m_2 \in \mathbb{Z}_p$ and $C_i = \mathrm{TGSW}_s(m_i) = Z_i + m_i \cdot \mathrm{G}^T$, we will verified that $C_1 \boxtimes C_2 = \mathrm{TGSW}_s(m_1 \cdot m_2)$.

*Proof.* We define $C_3 := C1 \boxtimes C_2$, then we got

$$\begin{aligned} C_3 = C_1 \boxtimes C_2 &= \mathrm{G}^{-1}(C_2) \cdot C_1 = \mathrm{G}^{-1}(C_2) \cdot (Z_1 + m_1.\mathrm{G}^T) \\ &= \mathrm{G}^{-1}(C_2) \cdot Z_1 + (\mathrm{G}^{-1}(C_2).m_1) \cdot \mathrm{G}^T \end{aligned} \tag{3}$$

Then, with $\varepsilon_2 = \mathrm{G}^{-1}(C_2).\mathrm{G}^T - C_2$ that corresponds to the error of the gadget transformation, we have

$$\begin{aligned} C_3 &= \mathrm{G}^{-1}(C_2) \cdot Z_1 + m_1.(C_2 + \varepsilon_2) \\ &= \mathrm{G}^{-1}(C_2) \cdot Z_1 + (m_1.m_2).\mathrm{G}^T + m_1.Z_2 + m_1.\varepsilon_2 \end{aligned} \tag{4}$$

If the noise coming from the terms $m_1.Z_2$ and $m_1.\varepsilon_2$ stay relatively 'small' then

$$C_3 = Z_3 + (m1.m2) \cdot \mathrm{G}^T$$

for $Z_3 = \mathrm{TGSW}_s(0) = \mathrm{G}^{-1}(C_2) \cdot Z_1$.

□

In the proof, we use the fact that, without considering the size of the noise, the sum of two encryption TLWE(0) stays an encryption TLWE(0) and if we multiply a TLWE(0) by a constant, it also stays a TLWE(0). With those two facts, we can deduce that

$$\begin{pmatrix} \mathrm{TLWE}_s(0) \\ \mathrm{TLWE}_s(0) \\ \vdots \\ \mathrm{TLWE}_s(0) \end{pmatrix} \times A = \begin{pmatrix} \mathrm{TLWE}_s(0) \\ \mathrm{TLWE}_s(0) \\ \vdots \\ \mathrm{TLWE}_s(0) \end{pmatrix}$$

Of course, the parameters change, but it's still a matrix whose rows are TLWE encryptions of 0.

Furthermore, we don't take too much into consideration, but the noise can grow quickly due to the term $m1.Z_2$ which multiplies the noise of each entry of $Z_2$ by $m_1$. We will now talk about the external product that output TLWE encryption smaller than TGSW one and the noise grows less than in the internal product.

**External Product :** Let $m_1 \in \mathbb{Z}_p$ and $\mu_1 \in \mathbb{T}_p$, the external product $m_1 \cdot \mu_1$ is well-defined and we want to define this external product to our ciphertext. To do that we define

$$\boxdot\colon \mathrm{TGSW} \times \mathrm{TLWE} \to \mathrm{TLWE} \tag{5}$$

$$(C_1, c_2) \mapsto C_1 \boxdot c_2 = \mathrm{G}^{-1}(c_2) \cdot C_1 \tag{6}$$

As we did before, we will prove that the result of $C_1 \boxdot c_2$ is an TLWE encryption.

*Proof.* Let $m_1 \in \mathbb{Z}_p$ and $\mu_2 \in \mathbb{T}_p$ such that $C_1 = \mathrm{TGSW}(m_1)$ and $c_2 = \mathrm{TLWE}(\mu_2)$. In more detail, we got:

$$C_1 = \mathrm{TGSW}(m_1) = Z_1 + m_1 \cdot \mathrm{G}^T \text{ and } c_2 \in \mathbb{T}_q^{n+1}$$

where

$$Z_1 = \begin{pmatrix} a_{1,1} & \dots & a_{1,n} & b_1 \\ a_{2,1} & \dots & a_{2,n} & b_2 \\ & \vdots & & \\ a_{(n+1)\ell,1} & \dots & a_{(n+1)\ell,n} & b_{(n+1)\ell} \end{pmatrix}$$

with

- $(a_{i,1}, \ldots, a_{i,n}) \xleftarrow{\mathcal{U}} \mathbb{T}_q^n$
- $b_i = \sum_{j=0}^{n} s_j \cdot a_{i,j} + (e_1)_i$ where $e_1$ is vector formed by all the error of the TLWE encryption in $Z_1$

and $c_2 = (a'_1, \ldots, a'_n, b')$ with

- $(a'_1, \ldots, a'_n) \xleftarrow{\mathcal{U}} \mathbb{T}_q^n$
- $b' = \sum_{i=0}^{n} s_i \cdot a'_i + \mu_2 + e_2$

Then,

$$\begin{aligned} c_3 := C_1 \boxdot c_2 &= \mathrm{G}^{-1}(c_2) \cdot C_1 \\ &= \mathrm{G}^{-1}(c_2) \cdot (Z_1 + m_1 \cdot \mathrm{G}^T) \\ &= \mathrm{G}^{-1}(c_2) \cdot Z_1 + m_1 \cdot (\mathrm{G}^{-1}(c_2) \cdot \mathrm{G}^T) \\ &= \mathrm{TLWE}_s(0) + m_1 \cdot c_2 \\ &= \mathrm{TLWE}_s(0) + m_1 \cdot \mathrm{TLWE}_s(\mu_2) \\ &= \mathrm{TLWE}_s(m_1 \cdot \mu_2) \end{aligned} \tag{7}$$

We used that $\mathrm{G}^{-1}(c_2) \cdot Z_1 = \mathrm{TLWE}_s(0)$ (sum and multiplication by a constant of $\mathrm{TLWE}_s(0)$) and $\mathrm{G}^{-1}(c_2) \cdot \mathrm{G}^T \approx c_2$ (and it's exact if $B^\ell = p$). □

### 3.7.2 TGLWE ciphertexts

**Important remark: (external multiplication)** $\mathbb{T}_{N,q}[X]$ is a $\mathbb{Z}_N[X]$-module (we use the fact that $\mathbb{T}_N[X]$ is a $\mathbb{Z}_N[X]$-module). So for $P \in \mathbb{Z}_N[X]$ and $Q \in \mathbb{T}_{N,q}[X]$, the multiplication $P \cdot Q$ is well defined and stays in $\mathbb{T}_{N,q}$. This external multiplication will be useful for TFHE but it is a demanding operation, requiring $N^2$ products in the Torus. The choice of $N$ enables efficient FFT to be performed, reducing this cost.

Again, the operations and underlying techniques developed for TLWE and TGSW extend to polynomials. Torus elements are replaced with torus polynomials. Addition and external multiplication are performed modulo $X^N + 1$. The same trick using a gadget matrix (over $\mathbb{T}_{N,q}[X]$) is used to control the noise growth.

**Additions** Let $\mu_1, \mu_2 \in \mathcal{P}_N[X]$ with respective ciphertexts $c_1 \leftarrow \mathrm{TGLWE}_s(\mu_1) = (a_1, \ldots, a_k, b) \in \mathbb{T}_{N,q}[X]^{k+1}$ and $c_2 \leftarrow \mathrm{TGLWE}_s(\mu_2) = (a'_1, \ldots, a'_k, b') \in \mathbb{T}_{N,q}[X]^{k+1}$. If $e_i$ is the noise present in $c_i$ $(i = 1, 2)$, then $c_3 := c_1 + c_2 = (a_1 + a'_1, \ldots, a_k + a'_k, b + b') \in \mathbb{T}_{N,q}[X]^{k+1}$ with $b + b' = \sum_{i=1}^{k} a_i \cdot s_i + \sum_{i=1}^{k} a'_i \cdot s_i + \mu_1 + \mu 2 + e1 + e2 = \sum_{i=1}^{k} (a_i + a'_i) \cdot s_i + (\mu_1 + \mu 2) + (e_1 + e_2)$. This is a valid TGLWE encryption of $\mu_3 := \mu_1 + \mu_2$ (in $\mathcal{P}_N[X]$), provided that the additive noise $e_3 := e_1 + e_2$ keeps small.

**Multiplication by a known polynomial** Let $\mu \in \mathcal{P}_N[X]$ and let $K \in \mathbb{Z} \subset \mathbb{Z}_N[X]$ (i.e., a degree 0 polynomial). Given the ciphertext $c \leftarrow \mathrm{TGLWE}_s(\mu)$, $c' := K \cdot c$ is a valid ciphertext of $\mu' = K \cdot \mu$ (in $\mathcal{P}_N[X]$), as long as the resulting noise remains "small."

Now if $K \in \mathbb{Z}_N[X]$, the same definition of $c'$ works because for $i \in [\![1, k]\!]$, $K \cdot a_i \in \mathbb{T}_{N,q}[X]$ (external product) and analogously $K \cdot e \in q^{-1}\mathbb{Z}_N[X] \subset \mathbb{T}_{N,q}[X]$ and $K \cdot \mu \in \mathbb{T}_{N,p}[X] \subset \mathbb{T}_{N,q}[X]$. Furthermore $(K \cdot a_i)_i$ and $(K \cdot e)$ will continue to behave according to the same random distributions we want them to. We then have that $c'$ is a valid ciphertext of $\mu' = K \cdot \mu$, still assuming that the noise is small.

**Multiplication of ciphertexts** As we did with TLWE ciphertexts we can define a new type of ciphertexts, the TGGSW ciphertexts, and also an homomorphic external multiplication between a TGGSW and a TGLWE, giving a TGLWE. The ideas behind are similar so we won't detail a lot.

**Definition 3.7** (Gadget matrix)**.** Let B a radix and $\ell \geq 1$ such that $B^l | q$. Adapting the dimension, we define the gadget matrix $G$ over $\mathbb{T}_{N,q}[X]$,

$$G \in \mathcal{M}_{k+1,(k+1)l}(\mathbb{T}_{N,q}[X]),$$

as

$$G^T := diag(g^T, \dots, g^T) = \begin{pmatrix} 1/B & & & & & \\ \vdots & & & & & \\ 1/B^l & & & & & \\ & 1/B & & & & \\ & \vdots & & & & \\ & 1/B^l & & & & \\ & & \ddots & & & \\ & & & & 1/B & \\ & & & & \vdots & \\ & & & & 1/B^l & \end{pmatrix}$$

with $g = \left(\frac{1}{B}, \dots, \frac{1}{B^\ell}\right) \in \mathbb{T}_q^\ell \subset \mathbb{T}_{N,q}[X]^\ell$.

This matrix allows us to go from an element in $\mathbb{Z}_N[X]^{(k+1)\ell}$ to an element in $\mathbb{T}_{N,q}[X]^{k+1}$ (multiplication to the right by the transpose of the gadget matrix).

The associated inverse transformation $G^{-1}(\cdot)$ flattens a vector of $(k+1)$ polynomials of $\mathbb{T}_{N,q}[X]$ into a vector of $(k+1)\ell$ polynomials in $\mathbb{Z}_N[X]$ with small coefficients (i.e., in the range $[-\lfloor B/2 \rfloor, \lceil B/2 \rceil]$). Again, it consits, for each element of the vector, in multiplying it by $B^\ell$ then rounding and finally "replacing" the modified element by its base-B decomposition (starting with the most significant coefficient).

Also, for any polynomial vector $p \in \mathbb{T}_{N,q}[X]^{k+1}$, it holds that:

$$G^{-1}(p) \cdot G^\top \approx p \quad \text{and} \quad G^{-1}(p) \text{ is "small."}$$

**TGGSW ciphertexts** Let $p = B^\ell$ and assume that $p \mid q$. Let also $s = (s_1, \dots, s_k) \in \mathbb{B}_N[X]^k$. The TGGSW encryption of $m \in \mathcal{P}_N[X]$ under the private key $s$ is defined as:

$$\text{TGGSW}_s(m) = Z + m \cdot G^\top \in \mathcal{M}_{(k+1)l,k+1}\mathbb{T}_{N,q}[X]$$

where

$$Z \leftarrow \begin{pmatrix} \text{TGLWE}_s(0) \\ \text{TGLWE}_s(0) \\ \vdots \\ \text{TGLWE}_s(0) \end{pmatrix} \quad \text{(a total of } (k+1)\ell \text{ rows)}$$

**External product of ciphertexts** Let $m_1 \in \mathcal{P}_N[X]$ and $\mu_2 \in \mathcal{P}_N[X]$ with their respective ciphertexts

$$C_1 \leftarrow \text{TGGSW}_s(m_1) \in \mathcal{M}_{(k+1)l,k+1}(\mathbb{T}_{N,q}[X]) \quad \text{and} \quad c_2 \leftarrow \text{TGLWE}_s(\mu_2) \in \mathbb{T}_{N,q}[X]^{k+1}.$$

The **external product** $\boxdot$ of a TGGSW ciphertext with a TGLWE ciphertext is defined as:

$$\boxdot : \text{TGGSW} \times \text{TGLWE} \to \text{TGLWE}, \quad (C_1, c_2) \mapsto C_1 \boxdot c_2 := G^{-1}(c_2) \cdot C_1.$$

We could prove as in the previous section that $c_3 := C_1 \boxdot c_2 \in \mathbb{T}_{N,q}[X]^{k+1}$ is a valid encryption of $\mu_3 := m_1 \cdot \mu_2 \in \mathcal{P}_N[X]$, provided that the rounding error from $G^{-1}(\cdot)$ and the multiplicative noise remain "small."

**Conclusion on ciphertexts multiplication** It's probably possible to go from a TGLWE ciphertext to a TGGSW one, allowing us to perform an external multiplication with another TGLWE and thus obtaining a TGLWE encryption of the product of the plaintexts. We know it's possible to change between different encryptions (without decrypting) but for this special case we didn't find information. Being able to do that would finally give us our holy homomorphic multiplication.

### 3.7.3 CMUX

We can use external multiplication in a "controlled multiplexer" (CMUX). Let $c_0 \leftarrow \text{TGLWE}_s(\mu_0)$ and $c_1 \leftarrow \text{TGLWE}_s(\mu_1)$ two ciphertexts. The CMUX operator uses a $\mathcal{C}_b \leftarrow \text{TGGSW}_s(b)$ encryption of a control bit to choose between $c_0$ and $c_1$.

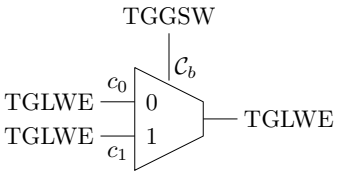


Figure 1: CMUX

Basically it can be seen as a boolean choice : if $\mathcal{C}_b = 0$ then it returns $c_0$ and if $C_b = 1$ then it returns $c_1$. In more "math" terms we can describe it as follows :

$$\begin{aligned} \text{CMUX}(\mathcal{C}_b, c_0, c_1) &:= C_b \boxdot (c_1 - c_0) + c_0 \\ &= \text{TGGSW}_s(b) \boxdot (\text{TGLWE}(\mu_1) - \text{TGLWE}(\mu_0)) + \text{TGLWE}(\mu_0) \\ &= \text{TGLWE}(b(\mu_1 - \mu_0) + \mu_0) \end{aligned} \tag{8}$$

## 3.8 Bootstrapping

### 3.8.1 Overview

The bootstrapping procedure was briefly described in the introduction 2.5. Gentry needed to reduce noise, and he found a way of doing so : you can reduce the noise by evaluating the decryption of the ciphertext using an homomorphically encrypted decryption key.

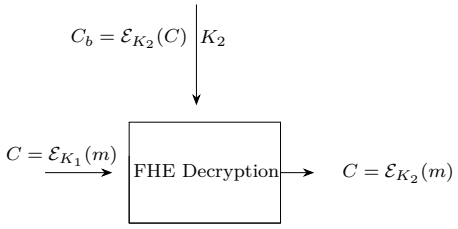


Figure 2: Bootstrapping

### 3.8.2 Description

Let's detail what we need to be able to do to bootstrap. Let $s = (s_1, \ldots, s_n) \in \mathbb{B}^n$, $\mu \in \mathbb{T}_p$ and $c = \text{TLWE}_s(\mu) = (a_1, \ldots, a_n, b) \in \mathbb{T}_q^{n+1}$ where

- $(a_1, \ldots, a_n) \xleftarrow{\mathcal{U}} \mathbb{T}_q^n$
- $b = \sum_{i=1}^{n} a_i \cdot s_i + \mu + e$ where $e$ is a small noise error

Our main goal here is to achieve to produce a TLWE ciphertext of the same plaintext $\mu$ but with an associated noise $e'$ smaller than $e$. To use the Gentry technique, we need to be able to decrypt the ciphertext with an encrypted key, and necessarily all the operations of the decryption need to be done homomorphically. In our case, to decrypt a TLWE ciphertext, we need to do two operations :

1. $\mu^* = b - \sum_{i=0}^{n} a_i \cdot s_i$
2. $\mu = \frac{\lfloor p\mu^* \rceil \mod p}{p} \in \mathbb{T}_p$

Since the first operation is linear, it's easy to do it homomorphically with LWE ciphertext if the $s_i$ are given encrypted, but the second one is more difficult. It's there that the polynomials and the TGGSW encryption will be crucial.

Let's give a little overview of why polynomials are so crucial and which operation we need to be able to 'round' homomorphically. First, we will see how to obtain a polynomial whose coefficient is the rounding number needed (**Polynomial rounding**). To be able to do that, we will need to 'rotate' the coefficient of the polynomial ($P \cdot X^{-j} \mod < X^N + 1 >$), and especially we need to do it homomorphically (**Blind Rotate**) (3.8.4). Once those two operations are done, we have to extract the coefficient needed from the polynomial as a TLWE ciphertext (**Sample extraction**)(3.8.5). The last operation is needed because at a moment another key appears and the TLWE ciphertext extracted is encrypted under another key than from the start, so we will need to be able to change the key and get the TLWE ciphertext encrypted under the original key (**Key switching**)(3.8.6).

**Remark :** Let's see what we call 'rotate' a polynomial. Since the set of plaintext is $\mathbb{T}_{N,p}[X]/ < X^N + 1 >$, we got the relation $X^{N+i} \equiv -X^i \mod X^N + 1$ for $i = 1, \dots, N-1$. Using this relation, if we take $P(X) = \sum_{i=0}^{N-1} p_i X^i \in \mathbb{T}_{N,p}[X]$ and $j \in [\![1, N-1]\!]$ then we get

$$P \cdot X^j = \sum_{i=0}^{j-1} -p_{i+N-j} X^i + \sum_{i=j}^{N-1} p_{i-j} X^i$$

As we can see, the coefficient of $P(X)$ 'rotate' (modulo a minus).

Furthermore, we also have the relation $X^{N+j} \cdot P(X) = -X^j \cdot P(X)$. If we combine the two preceding relations, we obtain that the constant of $X^{-j} \cdot P = X^{2N-j} \cdot P$ is $p_j$ if $j \in [\![0, N-1]\!]$ or $-p_j$ if $j \in [\![N, 2N-1]\!]$.

### 3.8.3 Rounding polynomial

Here, the idea is to construct a polynomial, called *rounding polynomial* and written $v(X) = \sum_{i=0}^{N-1} v_i X^i$, whose coefficients are all the possible values of $\lfloor p\mu^* \rceil \mod p$. Since $\mu^* \in \mathbb{T}_q$ then we can write $\mu^* = \frac{\bar{\mu^*}}{q}$ with $\bar{\mu^*} = \lfloor q\mu^* \rceil \mod q$. Then, $\bar{\mu^*}$ is in $[\![0, q-1]\!]$ and, if we suppose that $q < N$ (in practice it's not the case, but for now we suppose it), $v$ has more coefficients than the number of possible value for $\bar{\mu^*}$. Therefore, we can put all those values in the polynomial $v(X)$, more specifically, we define $v_j := \frac{\lfloor (j \cdot p)/q \rceil \mod p}{p} \in \mathbb{T}_p$. Finally, if we take $j = \bar{\mu^*}$ in the equation $X^{-j} \cdot v$ we have

$$\begin{aligned} X^{-j} \cdot v &= v_j + \dots \\ &= \frac{\lfloor (j \cdot p)/q \rceil \mod p}{p} + \dots \\ &= \frac{\lfloor (\bar{\mu^*} \cdot p)/q \rceil \mod p}{p} + \dots, \text{ since } j = \bar{\mu^*} \\ &= \frac{\lfloor \mu^* \cdot p \rceil \mod p}{p} \\ &= \mu + \dots \end{aligned} \tag{9}$$

and, in conclusion, the constant term of $X^{-j} \cdot v$ is $\mu$. The polynomial $v$ is known by everyone, but $\mu^*$ is encrypted, so we need to be able to do that operation homomorphically.

### 3.8.4 Blind-rotation

Before doing the operation $X^{-\bar{\mu^*}} \cdot v$, we need to be able to compute $-\bar{\mu^*}$. We recall that $\mu^* = b - \sum_{i=1}^n a_i \cdot s_i = \frac{\bar{\mu^*}}{q}$, this gives us $\bar{\mu^*} = q \cdot \mu^* = q \cdot b - \sum_{i=1}^n (a_i \cdot q) \cdot s_i$. Since $b, a_i \in \mathbb{T}_q$ then $bq, a_i q \in \mathbb{Z}_q = [\![0, q-1]\!] + \mathbb{Z}$ so we define $\bar{a_i} = \lfloor q a_i \rceil \mod q$ and $\bar{b} = \lfloor qb \rceil \mod q$. Hence,

$$-\bar{\mu^*} = -\bar{b} + \sum_{i=1}^n \bar{a_i} s_i \mod q$$

Then we can compute $-\bar{\mu^*}$, but it's important to notice that it's still encrypted since the $s_i$ are given encrypted. So we can compute $X^{-\bar{\mu^*}}$ directly but instead we will see a way to compute $X^{-\bar{\mu^*}} \cdot v$ without knowing $X^{-\bar{\mu^*}}$.

**test polynomial :** As said before, in fact, $N << q$ ($N \in \{2^{10}, 2^{11}, 2^{12}\}$, $q \in \{2^{32}, 2^{64}\}$). We will get over this issue in two steps:

- First, since we work in $Z_N[X]$, $X$ is of order $2N$, then we get $X^{-\bar{\mu^*}} \cdot v = X^{(-\bar{\mu^*}) \mod 2N} \cdot v$. Hence, we need to rescale $-\bar{\mu^*}$ modulo $2N$ and to do so, instead of starting from the relation $-\bar{\mu^*} = -\bar{b} + \sum_{i=1}^{n} \bar{a_i} s_i \mod q$ we use

$$-\tilde{\mu^*} = -\tilde{b} + \sum_{i=1}^{n} \tilde{a_i} s_i \mod 2N$$

  where $\tilde{b} = \lfloor 2Nb \rceil \mod 2N$ and $\tilde{a_i} = \lfloor 2Na_i \rceil \mod 2N$. This operation can add some noise.

- Secondly, since $v \in \mathbb{T}_{N,q}[X]$, $v$ has $N$ coefficients, $v$ can only store $N$ values for $\tilde{\mu^*}$. By ensuring that the first bit $\tilde{\mu^*}$ is set to 0, $\tilde{\mu^*}$ can take at most $N$ possible values.

By combining the two precedent points, we construct the *test polynomial* $v$ defined by

$$v = \sum_{i=0}^{N-1} v_i \cdot X^i \text{ with } v_i = \frac{\lfloor \frac{pj}{2N} \rceil \mod p}{p} \in \mathbb{T}_p$$

and the relation

$$X^{-\tilde{\mu^*}} \cdot v = X^{-\tilde{b} + \sum_{i=1}^{n} \tilde{a_i} s_i} \cdot v = X^{-\bar{\mu^*}} \cdot v = \mu + \dots$$

holds if we have that the noise added by the first operation is not too large and $-\tilde{\mu^*} \mod 2N \in [\![0, N-1]\!]$.

Now, we will find a way to compute $X^{-\tilde{\mu^*}} \cdot v$ homomorphically. To do so we will find a way to do it recursively by using CMux as an if-else gate. Let $q_j = X^{-\tilde{b} + \sum_{i=1}^{j} s_i \cdot \tilde{a_i}} \cdot v$, we get

$$\begin{aligned} q_j &= (X^{-\tilde{b} + \sum_{i=1}^{j-1} s_i \cdot \tilde{a_i}} X^{s_j \cdot \tilde{a_j}}) \cdot v = X^{s_j \cdot \tilde{a_j}} (X^{-\tilde{b} + \sum_{i=1}^{j-1} s_i \cdot \tilde{a_i}} \cdot v) = X^{s_j \cdot \tilde{a_j}} \cdot q_{j-1} \\ &= \begin{cases} q_{j-1} \text{ if } s_j = 0 \\ X^{\tilde{a_j}} \cdot q_{j-1} \text{ if } s_j = 1 \end{cases} \end{aligned} \tag{10}$$

Then, by using Cmux 3.7.3 as an if-else gate, we can compute $X^{-\tilde{b} + \sum_{i=1}^{n} s_i \cdot \tilde{a_i}} \cdot v = X^{-\tilde{\mu^*}} \cdot v$ starting from $q_0 = X^{-\tilde{b}} \cdot v$. Now that we have everything needed, we write the algorithm on the clear and over the encrypted data. Let $\mathbf{s'} \in \mathbb{B}_N[X]^{k+1}$ be the secret key used to encrypt the bootstrapping keys that we suppose given. We denote the bootstrapping keys as bsk with $\text{bsk}[j] = TGGSW_{s'}(s_j) \in \mathcal{M}_{(k+1)\ell,(k+1)}(\mathbb{T}_{N,q}[X])$ for $j \in [\![1, n]\!]$ and then we have :

| Algorithm in the clear | Algorithm over the encrypted data |
|---|---|
| $q_0 \leftarrow X^{-\tilde{b}} \cdot v$<br>**for** $j \leftarrow 1$ to $n$ **do**<br>$q_j \leftarrow \begin{cases} q_{j-1} \text{ if } s_j = 0 \\ X^{\tilde{a_j}} \cdot q_{j-1} \text{ if } s_j = 1 \end{cases}$<br>**end for**<br>**return** $q_n$ | $c'_0 \leftarrow X^{-\tilde{b}} \cdot \text{TGLWE}_{\mathbf{s'}}(v)$<br>**for** $j \leftarrow 1$ to $n$ **do**<br>$c'_j \leftarrow \text{CMUX}(bsk[j], c'_{j-1}, X^{\tilde{a}_j} \cdot c'_{j-1})$<br>**end for**<br>**return** $c'_n$ |

First, we recall that the output of CMux is a TGLWE encryption then $c' := c'_n$ is a TGLWE encryption of $q_n = X^{-\tilde{b} - \sum_{i=1}^{n} s_i \cdot \tilde{a_i}} \cdot v$ with the key $\mathbf{s'}$. Secondly, the vector $(0, \dots, 0, v) \in \mathbb{T}_{N,q}[X]^{k+1}$ is

a valid TGLWE encryption of $v$, called trivial TGLWE encryption, then we can take $c'_0 = TLWE_s(\mu) \in \mathbb{T}_q^{n+1}$.

Let's take a step back and see what we get. We start with a ciphertext $c = TLWE(\mu)_s$ with $s = (s_1, \ldots, s_n) \in \mathbb{B}^n$ and the bootstraping key $\text{bsk}[j] = TGGSW_{s'}(s_j)$. Then, we construct the TGLWE ciphertext $c' = \text{TGLWE}(X^{-\tilde{\mu^*}} \cdot v) = \text{TGLWE}(\mu + \ldots)$ with $v = \sum_{j=0}^{N-1} \frac{\lfloor \frac{pj}{2N} \rceil \mod p}{p} \cdot X^j$ the *test polynomial* in only two step :

1. Define $c = (0, \ldots, 0, v)$ and $\bar{c} = (\bar{a_1}, \ldots, \bar{a_n}, \bar{b})$
2. do
$$\begin{cases} c'_0 \leftarrow X^{-\bar{b}} \cdot c \\ c'_j \leftarrow \text{CMux}(\text{bsk}[j], c'_{j-1}, X^{\bar{a_j}} \cdot c'_{j-1}) \text{ for } j \in [\![1, n]\!] \end{cases} \tag{11}$$
and set $c' := c'_n$.

We write $c' \leftarrow \text{BlindRotate}_{\text{bsk}}(c, c')$ where bsk=(bsk[1],...,bsk[n]).

### 3.8.5 Sample extract

We want now to extract the constant coefficient of $c' = \text{TGLWE}_{s'}(\mu + \ldots) = (a'_1, \ldots, a'_k, b') \in \mathbb{T}_{N,q}[X]$ as a TLWE ciphertext under a secret key $\mathbf{s}'$ that we will define later. To do that, we will show that the constant coefficient of $b' \in \mathbb{T}_{N,q}[X]$ is a TLWE encryption of $\mu$ with some parameter that we can construct only using $c'$. We now give the details of the procedure.

Let $s' = (s'_1, \ldots, s'_k) \in \mathbb{B}_N[X]^k$ and $c' = \text{TGLWE}_{s'}(P_\mu) = (a'_1, \ldots, a'_k, b') \in \mathbb{T}_{N,q}[X]$ where $s'_j = \sum_{i=0}^{N-1} s_i'^{(j)} \cdot X^i$ for $j \in [\![1, k]\!]$, $a'_j = \sum_{i=0}^{N-1} a_i'^{(j)} \cdot X^i$ and $P_\mu = X^{-\bar{\mu}^*} \cdot v = \mu + \ldots \in \mathbb{T}_{N,q}[X]$. We define also $e = \sum_{i=0}^{N-1} e_i \cdot X^i$ such that $b' = \sum_{i=0}^{k} s'_j \cdot a'_j + P_\mu + e$. If we expand the polynomial $b$, we get

$$\begin{aligned} b' &= \sum_{i=0}^{N-1} b'_i \cdot X^i \\ &= \sum_{j=0}^{k} \Big( \Big( \sum_{i=1}^{N-1} s_i'^{(j)} \cdot X^i \Big) \cdot \Big( \sum_{i=1}^{N-1} a_i'^{(j)} \cdot X^i \Big) \Big) + P_\mu + e \end{aligned} \tag{12}$$

and, if we look at its constant coefficient, we obtain

$$\begin{aligned} b'_0 &= s_0'^{(j)} \cdot a_0'^{(j)} - \sum_{j=1}^{k} \sum_{i=1}^{N-1} s_i'^{(j)} \cdot a_{N-i}'^{(j)} + \mu + e_0 \\ &= < (s_0'^{(1)}, s_1'^{(2)}, \ldots, s_{N-1}'^{(1)}, \ldots, s_0'^{(k)}, \ldots, s_{N-1}'^{(k)}), \\ &\quad (a_0'^{(1)}, -a_{N-1}'^{(1)}, \ldots, -a_1'^{(1)}, \ldots, a_0'^{(k)}, -a_1'^{(k)}, \ldots, -a_1'^{(k)}) > \\ &\quad + \mu + e. \end{aligned} \tag{13}$$

So, if we define $\mathbf{s}' = (s_0'^{(1)}, s_1'^{(2)}, \ldots, s_{N-1}'^{(1)}, \ldots, s_0'^{(k)}, \ldots, s_{N-1}'^{(k)}) \in \mathbb{B}^{kN}$

and $\mathbf{a}' = (a_0'^{(1)}, -a_{N-1}'^{(1)}, \ldots, -a_1'^{(1)}, \ldots, a_0'^{(k)}, -a_1'^{(k)}, \ldots, -a_1'^{(k)}) \in \mathbb{T}_q^{kN}$, then the vector $\mathbf{c}' = (\mathbf{a}', b'_0) \in \mathbb{T}_q^{kN+1}$ can be viewed as a TLWE encription of $\mu$ under the secret key $\mathbf{s}'$. It's important to notice this is a 'fresh' TLWE ciphertext, i.e, the noise of this TLWE ciphertext is small as when we encrypt a plaintext. We write $\mathbf{s}' \leftarrow \text{Recode}(s')$ and $\mathbf{c}' \leftarrow \text{SampleExtract(c')}$.

### 3.8.6 Key switching

Before we see the algorithm to switch from $\mathbf{s}'$ to $s$ for our TLWE ciphertext, we will present an other idea to skip this part. If it's safe to encrypt the secret key with itself, then the bootstraping key become $\text{bkj}[j] = TGGSW_s(s_j)$ and $s' = s$. Thus gives us that the TLWE construct before is a 'fresh' TLWE ciphertext of $\mu$ under the secret key $s$. There is still the problem of the dimension of the ciphertext,

recall $c = \text{TLWE}_s(\mu) \in \mathbb{T}_q^{n+1}$ and $\mathbf{c}' = \text{TLWE}_{s'}(\mu) \in \mathbb{T}_q^{kN+1}$. It's possible to get through this problem but we won't do it here.

Now, we will focus on the key-switching algorithm. First, we will need a key-switching key that is define by the TLWE encryption of the bit $s'$ with the secret key $s$, in details we define $\text{ksk}(i,j) = \text{TLWE}_s(s'_i \cdot B^{-j})$ for $i \in [\![1, kN]\!]$ and $j \in [\![1, l]\!]$ where $B$ and $l$ are parameter that define a gadget matrix. We recall that $c' = (a'_1, \dots, a'_{kN}, b') \in \mathbb{T}_q^{kN+1} = \text{TLWE}_{\mathbf{s}'}(\mu)$ with $\mathbf{s}' = (s'_1, \dots, s'_{kN}) \in \mathbb{B}^{kN}$. Then the ciphertext

$$\mathbf{c}'' = (0, \dots, 0, b') - \sum_{i=1}^{kN} \sum_{j=1}^{l} (\bar{a'_i})_j \cdot \text{ksk}(i,j)$$

where

$$((\bar{a'_i})_1, \dots, (\bar{a'_i})_l) = g^{-1}(a'_i) \text{ with } (\bar{a'_i})_j \in [-\left\lfloor \frac{B}{2} \right\rfloor, \left\lceil \frac{B}{2} \right\rceil]$$

is a valid TLWE encryption of $\mu$ with the secret key $s \in \mathbb{B}^n$ (the original one).

We write $c'' \leftarrow \text{KeySwitch}_{\text{ksk}}(\mathbf{c'})$

This result can seem confusing, let's see why this works.

*Proof.* With the gadget decomposition 3.6 we get that

$$g^{-1}(a'_i) \cdots g^T = \sum_{j=1}^{l} (\bar{a'_i})_j B^{-j} = a'_i + \epsilon_i$$

with $\epsilon_i$ the rounding error. We recall that the computation of $g^{-1}(a'_i) \cdots g^T$ might be inexact if $B^l \neq p$ that where the rounding error comes from. Thus,

$$\begin{aligned} \sum_{j=1}^{l} (\bar{a'_i})_j \cdot \text{ksk}(i,j) &= \sum_{j=1}^{l} (\bar{a'_i})_j \cdot \text{TLWE}_s(s'_i \cdot B^{-j}) \\ &= \text{TLWE}_s(\sum_{j=1}^{l} (\bar{a'_i})_j \cdot s'_i \cdot B^{-j}) \\ &= \text{TLWE}_s(s'_i \cdot \sum_{j=1}^{l} (\bar{a'_i})_j \cdot B^{-j}) \\ &= \text{TLWE}_s(s'_i \cdot (a'_i + \epsilon_i)) \end{aligned} \tag{14}$$

Furthermore, $(0, \dots, 0, b')$ is a valid TLWE encryption of $b'$. We denote $e'$ the noise present in $c'$, then

$$\begin{aligned} c'' :&= \text{TLWE}_s(b' - \sum_{i=1}^{kN} s'_i \cdot (a'_i + \epsilon_i)) \\ &= \text{TLWE}_s(\mu + e' + \sum_{j=1}^{kN} s'_i \cdot \epsilon_i) \text{ , recall : } b' = \sum_{i=1}^{kN} a'_i \cdot s'_i + \mu + e' \end{aligned} \tag{15}$$

and $c''$ decrypts to $\mu$ if the error $e'' := e' + \sum_{i=1}^{kN} s'_i \cdot \epsilon_i$ stays 'small'. □

This procedure looks like the bootstraping in the idea but there is 2 main difference : the bootstraping is slow but reduce the noise meanwhile switching-key is fast but increase the noise. But the noise doesn't increase too much and that why TFHE is a FHE.

### 3.8.7 Final result

We have now everything needed to build an algorithm of bootstrapping for TFHE. Starting from a TLWE ciphertext $c = \text{TLWE}_s(\mu) \in \mathbb{T}_q^{n+1}$ with $s = (s_1, \dots, s_n) \in \mathbb{B}^n$, we get the following algorithm:

1. $c' \leftarrow \text{BlindRotate}_{bsk}(c, \tilde{c})$ with :
   - $c = (0, \dots, 0, v) \in \mathbb{T}_n^q[X]^{k+1}$ with $v := \sum_{j=0}^{N-1} \frac{\lfloor p_j/(2N) \pmod p \rceil}{p} X^j$

- $\tilde{c} = \lfloor c2N \rceil \in (\mathbb{Z}/2N\mathbb{Z})^{n+1}$
- $\text{bsk} = (\text{bsk}[j])_{1 \leq j \leq n}$ with : $\begin{cases} \text{bsk}[j] \leftarrow \text{TGGSW}_{s'}(s_j) \in \mathcal{M}_{(k+1)l,k+1}\mathbb{T}_{N,q}[X] \\ s' = (s'_1, \ldots, s'_k) \in \mathbb{B}_n[X]^k \end{cases}$

2. $c' \leftarrow \text{SampleExtract}(c') \in \mathbb{T}_q^{kN+1}$
3. $c'' \leftarrow \text{KeySwitch}_{ksk}(c')$ with :
   - $s' = (s'_1, \ldots, s'_{kN}) \leftarrow \text{Recode}(s') \in \mathbb{B}^{KN}$
   - $\text{ksk}[i,j] \leftarrow \text{TLWE}_s(s'_i \cdot B^{-j}) \in \mathbb{T}_q^{n+1}$
   - $\text{ksk} = (\text{ksk}[i,j])_{\substack{1 \leq i \leq kN \\ 1 \leq j \leq N}}$

### 3.9 Programmable bootstrapping

The first idea that we used to bootstrap was that $X^{-j} \cdot v = v_j + \ldots$ and the definition of $v_j$. Now, we will see what happens if we change the definition of $v_j = \frac{\lfloor \frac{pj}{2N} \rceil \mod p}{p} \in \mathbb{T}_p$. Let $f : \mathbb{T}_p \longrightarrow \mathbb{T}_p$, if we define $v_j$ as $f(\frac{\lfloor \frac{pj}{2N} \rceil \mod p}{p})$ then $X^{-j} \cdot v = v_j + \ldots$ will now have as a constant coefficient $f(\mu)$. Then, by applying the same procedure has before the 'fresh' TLWE encryption will be $\text{TLWE}_s(f(\mu))$ and not $\text{TLWE}_s(\mu)$. In fact, the bootstrapping in the last part was with $f = id$.

Moreover, if we consider a function $f$ such that $f(\mu + \frac{1}{2}) = -f(\mu)$, we say that $f$ is negacyclic, then we don't need to restrict the possible value of $-\tilde{\mu}$. For example, the function sign over the torus is negacyclic.

## 4 Computation time

To try out TFHE we used the Concrete python library made by Zama. It allows to create an fhe compiler for a chosen function $f$ (that must be coded in a specific way), and then, specifying an inputset, to create a circuit. This circuit is finally used to encrypt, decrypt, and evaluate the function (client-side and server-side can be separated). Everything is encapsulated so we didn't have to think a lot about the implementation but it allowed us to make several little tests to see the time taken for each operation (addition 3, multiplication 4, Lookup Table, comparisons, bit operations). To be honest it seems very long, even for operations that should be very fast : 0.6 seconds for a table lookup or for a bitwise "And" for instance. Nevertheless, it seems like it can be optimized, but the library is hard to use. In particular for some functions available with numpy it seems more efficient, which is coherent because numpy is used in Concrete implementation. It is also important to remember that python is not made to be fast, it is an interpreted language.

We then tested the evolution of the time taken by the fhe compiler to process when the number of operation increased.

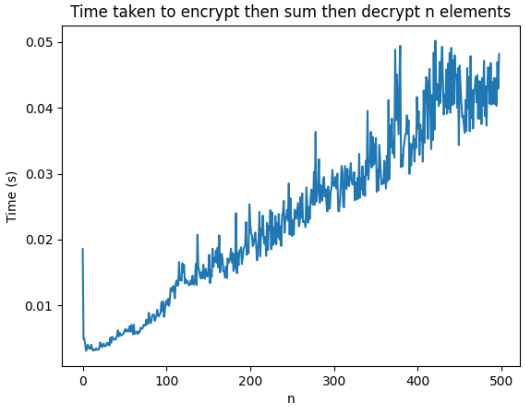


Figure 3: Additions times

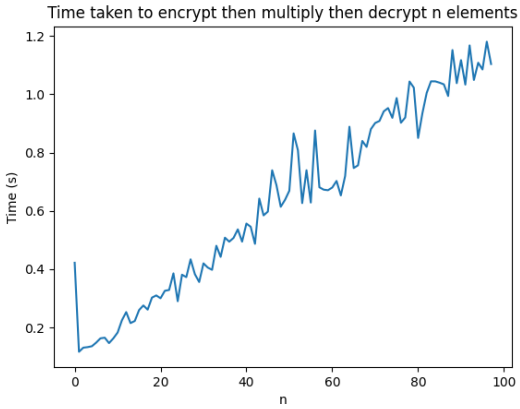


Figure 4: Multiplication times

# 5 Credits

Marc Joye's guide [3] helped us a lot to describe all the steps in TFHE, we relied on it a lot but tried to explain with our own words when possible. We tried to add a more intuitive idea at each step.

# References

[1] Marcolla, Chiara; Sucasas, Victor; Manzano, Marc; Bassoli, Riccardo; Fitzek, Frank H.P.; Aaraj, Najwa (2022) *Survey on Fully Homomorphic Encryption, Theory, and Applications*

[2] Léo Ducas; Daniele Micciancio *FHEW: Bootstrapping Homomorphic Encryption in less than a second*

[3] Marc Joye, Zama France; *Guide to Fully Homomorphic Encryption over the [Discretized] Torus*

[4] Alice Silverberg; *Fully Homomorphic Encryption For Mathematicians*

[5] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva and Malika Izabach; *Fast Fully Homomorphic Encryption over the Torus*